1 Symbol tables in MIPS

Recall that the address of a label is given by 4 times the number of non-blank lines preceding it in the program, where a blank line is a line not containing an instruction or .word directive. The symbol table for a MIPS assembly program is a list of labels in that program and their associated addresses.

Construct the symbol table for the following MIPS assembly program.

begin:
label: beq $0, $0, after
jr $4

after:
sw $31, 16($0)
lis $4
abc0: abc1: .word after

loadStore:
lw $20, 4($0)
sw $20, 28($0)

end:

2 Error-checking MIPS programs

Identify the errors in the following assembly language program. You might find it helpful to think of what a valid line ought to look like and then try to identify lines that do not match your expectations, rather than trying to think of all of the ways that lines can be invalid.

label: label: .word label
.word ; 0
.word aaaaa
.word 1 2 3
.word 2147483648 abcde:
.word ,
3 Bitwise operations in MIPS

Bitwise operations enable precise manipulation of bit patterns. The bitwise operations that will be useful in this course are the following

- And (&), which produces the word where bits are 1 if they were 1 in both inputs:
  
  \[
  \begin{array}{c}
  1011 \\
  \& 1101 \\
  \hline
  1001
  \end{array}
  \]

- Inclusive Or (|), which produces the word where bits are 1 if they were 1 in either input:

  \[
  \begin{array}{c}
  1010 \\
  | 1001 \\
  \hline
  1011
  \end{array}
  \]

- Left shift (<<), which removes the number of bits given on the right hand side from the beginning of the number on the left hand side, and adds that many zeros on the end of that number.

  \[
  \begin{array}{c}
  0110 \\
  \ll 2 \\
  \hline
  1000
  \end{array}
  \]

- Right shift (>>, which works the same way as left shift except that bits are removed from the end and added to the beginning.

  \[
  \begin{array}{c}
  0110 \\
  \gg 2 \\
  \hline
  0001
  \end{array}
  \]

Note that the >> operator in C may work differently on signed integers, so you should be careful to always use unsigned integers when right shifting.

Note that in Racket these operations are bitwise-and, bitwise-ior, and arithmetic-shift with positive and negative shift values respectively.

1. Assume unsigned 4 bit integers. What should each of the following computations produce?

   (a) 3 & 5
   (b) 3 | 5
   (c) 3 << 2
   (d) 3 >> 2
   (e) 13 << 2
   (f) 13 >> 2

2. Give an example of where each bitwise operation might be useful when writing an assembler.
4 Binary output

1. Write pseudocode for a function called `output_word` that takes a 32-bit integer as input and outputs each of its four bytes to standard output. You can assume a function called `output_byte` is available.

Assume the `output_byte` function takes an integer as input. If the integer is small enough to fit in a byte it outputs that byte to standard output; otherwise it produces an error. The `output_byte` function corresponds to:

- `putchar` in C/C++.
- `write-byte` in Racket.
- `System.out.write` in Scala.

2. How would you use this above function (in conjunction with the symbol table) to assemble the `.word foo` directive, where foo is a label in an assembly program?